

# A Genetic Algorithm for Software Design Migration from Structured to Object Oriented Paradigm

Md. Selim

Institute of Information Technology  
University of Dhaka, Bangladesh  
selim.iitdu@gmail.com

Saeed Siddik

Institute of Information Technology  
University of Dhaka, Bangladesh  
siddik.saeed@gmail.com

Alim Ul Gias

Institute of Information Technology  
University of Dhaka, Bangladesh  
alimulgias@gmail.com

M. Abdullah-Al-Wadud

Department of Industrial and Management Engineering  
Hankuk University of Foreign Studies, South Korea  
wadud@hufs.ac.kr

Shah Mostafa Khaled

Institute of Information Technology  
University of Dhaka, Bangladesh  
khaled@univdhaka.edu

**Abstract:** The potential benefit of migrating software design from Structured to Object Oriented Paradigm is manifold including modularity, manageability and extendability. This design migration should be automated as it will reduce the time required in manual process. Our previous work has addressed this issue in terms of optimal graph clustering problem formulated by a quadratic Integer Program (IP). However, it has been realized that solution to the IP is computationally hard and thus heuristic based methods are required to get a near optimal solution. This paper presents a Genetic Algorithm (GA) for optimal clustering with an objective of maximizing intra-cluster edges whereas minimizing the inter-cluster ones. The proposed algorithm relies on fitness based parent selection and cross-overing cluster elements to reach an optimal solution step by step. The scheme was implemented and tested against a set of real and synthetic data. The experimental results show that GA outperforms our previous works based on Greedy and Monte Carlo approaches by 40% and 49.5%.

**Key-Words:** Software Design Migration, Optimal Graph Clustering, Genetic Algorithm

## 1 Introduction

Software design migration from Structured to Object Oriented paradigm is essential for large legacy software [1] due to its lack of modularity, manageability and extendability. A possible way of shifting the paradigm could be re-designing the whole product from the scratch or manual design migration which could be error-prone and time consuming. An automated Structured to Object Oriented paradigm migration could reduce those errors and time consumption hence, motivating industries to adopt the procedure.

This scenario has been represented as an optimal graph clustering problem. It has been formalized in our previous work [3] with  $G(V, E)$  as the underlying undirected graph of a call graph with  $V$  and  $E$  as the set of vertices and edges respectively,  $n = |V|$ ,  $m = |E|$ .

The problem of maximizing intra-cluster edges, minimizing inter-cluster edges, and maximizing the number of clusters is used as an index to measure quality of a clustering. The matrix is referred to as  $Kal(\kappa)$  in the rest of this paper:

$$\kappa = \sum_i x_i - \sum_i y_i + \sum_j |C_j| \quad (1)$$

Here  $x_i, y_i \in \{0, 1\}$ ,  $i = 1, 2, \dots, m$  refers to the intra-cluster edges,  $x_i = 1$  if  $x_i$  is intra-cluster;  $y_i = 1$  if  $y_i$  is inter-cluster.  $C_j, j = 1, 2, \dots, n$  represent cluster heads.  $C_j \in \{0, 1\}$  if vertex  $j \in V$  is the head of a cluster.

Selim [4] proved the problem to be a computationally hard optimization problem. The optimal solution to such problems cannot be found in polynomial time, and therefore search for solution to the problem has to rely on approximation or heuristics.

This paper introduces a Genetic Algorithm for optimal graph clustering that focuses on maximizing and minimizing the intra and inter cluster edges respectively. The fitness of each cluster is significantly depended on it's

intra cluster edges. Based on the fitness of each individual clusters within a clustering, pairs are formed. A cross-over takes place within those generated pairs to exchange the vertices. Mutation within a cluster may take place based on a probability distribution.

The algorithm was implemented where a clustering scheme  $C$  yielded by a greedy algorithm [3] was used as an initial seed. That implementation was assessed based on 3 data instances that includes both real life and synthetically generated ones. The results show that the genetic algorithm produced better results in terms of all metrics used in [3] which include clustering coefficient ( $\Psi$ ) [13], characteristic path length ( $\chi$ ) [13] and Kal ( $\kappa$ ) index.

Rest of the paper is organized as follows: Section 2 reviews the research done on SP to OOP design migration, graph clustering, and presents matrices to measure clustering quality. Section 3 presents the proposed genetic algorithm approach, Section 4 presents the data and experimental results, Section 5 concludes the paper.

## 2 Related Work

State of the art works regarding software design migration includes automatic migration from code to design [5], hierarchical clustering research in the context of software architecture recovery and modularization [6], architectural comparison of commercial software and scientific research software [7] and empirical approach for migrating from Structured Programming Code to Object Oriented Design [8].

Franti et al. [9] used variations of Genetic algorithm approaches for solving large scale clustering problem. They introduced three new efficient crossover techniques that are the hybrid outcome of genetic algorithm and k-means algorithm. Their proposed techniques are based on k-dimensional Euclidean distances.

A self adaptive genetic algorithm is proposed in [10] for cluster analysis that associates a set of parameters with each cluster and these parameters update by crossover and mutation. Wang et al. [11] presented a fuzzy genetic algorithm for cluster analysis with c-means clustering algorithm. They used genetic algorithm for minimizing the risk of trapping in local minimum. Hruschka et al. [12] proposed a genetic algorithm for finding a right number of clusters. They also used an encoding schema for determining chromosome, and Silhouette method is used for validating cluster data. Maulik et al. [14] done a comparative study on k-means and genetic algorithm for cluster finding. The used n-dimensional for searching cluster centers.

Recently, Saeed et al. modeled structured to object oriented design migration as a optimal graph clustering problem which is realized as computationally hard [3, 4]. They developed certain heuristic algorithms based on Monte Carlo and Greedy approaches and formulated the Kal ( $\kappa$ ) index for measuring the quality of a cluster. Moreover the clustering coefficient ( $\Psi$ ) and characteristic path length ( $\chi$ ) was used for assessing the quality.

Clustering Coefficient (CC) [13, 15] is a measure of degree to which vertices in a graph tend to cluster together. Local clustering coefficient can be used to measure CC ( $\Psi$ ) index where the local clustering coefficient of a vertex quantifies how close its neighbors are to being a complete graph. Suppose, a graph  $G = (V, E)$  consists of a set of vertices  $V$  and a set of edges  $E$ . If an edge  $e_{ij}$  connects vertex  $v_i, v_j$ , the neighborhood  $N_i$  for the vertex  $v_i$  is defined as its immediately connected neighbors:  $N_i = \{v_j : e_{ij} \in E \cap e_{ij} \in E\}$ . Clustering Coefficient  $\Psi$  of an undirected graph is defined as-

$$\Psi = \frac{1}{N} \sum_{i=1}^N \Psi_i \quad (2)$$

$$\text{where } \Psi_i = \frac{2|\{e_{ij} : v_j, v_k \in N_i, e_{jk} \in E\}|}{K_i(K_i - 1)}$$

Equation 2  $\Psi_i$  denotes the clustering coefficient of node  $i$  and  $k_i$  is number of vertices connected to vertex  $i$ , and  $n_i$  is actual number of edges within  $k_i$  adjacent vertices.

Characteristics Path Length (CPL) [13, 15] is the distance between pairs of vertices in a connected undirected graph [15]. Let  $d(v_i, v_j)$  denote the shortest distance between vertices  $v_i$  and  $v_j$ , where  $\{v_1, v_2\} \in V$  in an unweighed undirected graph  $G$ . If  $v_1 = v_2$  or  $v_2$  cannot be reached from  $v_1$  then  $d(v_i, v_j) = 0$ , otherwise  $d(v_i, v_j) = 1$ . Based on these definitions, Characteristics Path Length  $\chi$  of an undirected graph can defined as-

$$\chi = \frac{1}{N(N-1)} \cdot \sum_{i \neq j} d(v_i, v_j) \quad (3)$$

Review of the state of the art works show that software design migration using graph clustering did not receive high attention from the researchers. However, the scope of addressing the issue has broadened as it have been modeled in [3]. Different meta heuristic based algorithms can be used utilizing the model to reach an optimal solution to the problem.

### 3 Proposed Genetic Algorithm for Design Migration

The Genetic Algorithm based meta-heuristic approach presented in this section has the underlying undirected graph  $G(V, E)$  of a call graph as the input. It produces a clustering scheme  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$  with  $k$  clusters of vertices  $v_i \in V$ , with  $\cup_{i=1..k} C_i = V$  and  $\cap_{i=1..k} C_i = \emptyset$ , as a clue to a modularized object oriented design. This clusters represent the underlying potential classes and/or interfaces in the future object oriented design.

---

#### Algorithm 1 Genetic Algorithm for Graph Clustering

---

**Input:** Call Graph  $G(V, E)$

**Output:** Clustering  $\mathcal{C}$ , Clustering Coefficient ( $\Psi$ ), Characteristics Path Length ( $\chi$ ),  $Kal(\kappa)$

```

1: Begin
2: Randomly associate a unique integer order  $\tilde{o}(v)$  to all  $v \in V$  so that  $\cup \tilde{o}(v) \subset \mathbb{Z}$  and  $\cap \tilde{o}(v) = \emptyset$ 
3:  $\mathcal{C}_{init} \leftarrow GreedyClustering(G)$ 
4:  $\mathcal{C} \leftarrow \mathcal{C}_{init}$ 
5: repeat
6:    $\mathcal{C}_{tmp} \leftarrow \mathcal{C}$ 
7:    $F \leftarrow FitnessCalculation(\mathcal{C}_{tmp}, G)$ 
8:    $\mathcal{P} \leftarrow ParentSelection(\mathcal{C}_{tmp}, F)$ 
9:    $\mathcal{C}_{tmp} \leftarrow CrossOver(\mathcal{C}_{tmp}, \mathcal{P}, \tilde{O})$ 
10:   $\mathcal{C}_{tmp} \leftarrow Mutation(\mathcal{C}_{tmp}, \tilde{O})$ 
11:  if  $\kappa_{\mathcal{C}_{tmp}} \geq \kappa_{\mathcal{C}}$  then
12:     $\mathcal{C} \leftarrow \mathcal{C}_{tmp}$ 
13:  end if
14: until  $\kappa_{\mathcal{C}_{tmp}}$  does not improve for  $t$  consecutive iterations
15: Calculate  $\Psi$ ,  $\chi$ ,  $\kappa$  using  $\mathcal{C}$  and Eq. (2), (3) and (1)
16: End

```

---



---

#### Algorithm 2 GreedyClustering [3]

---

**Input:** Call Graph  $G(V, E)$

**Output:** Clustering  $\mathcal{C}$

```

1: Begin
2: Fix initial number of clusters  $\mathcal{C}_{i=1, \dots, n}$  to  $n = \sqrt{|V|}$ 
3: Pick unique vertex  $v_i \in V$  in decreasing order of vertex degree and a make one-to-one correspondence assignment of  $v_i$  to  $\mathcal{C}_j$ , where  $i, j = 1, 2, 3 \dots n$ 
4: for each edge  $e \in E$  do
5:   Assume  $v_1$  and  $v_2$  be the two end points of  $e$ 
6:   if  $v_1 \in \mathcal{C}_i$  and  $v_2$  unassigned to any cluster then
7:      $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v_2\}$ 
8:   else if  $v_1$  unassigned to any cluster and  $v_2 \in \mathcal{C}_j$  then
9:      $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v_1\}$ 
10:  end if
11: end for
12: End

```

---

The proposed scheme for optimal graph clustering using genetic algorithm is presented in Algorithm 1. The initial seed for the algorithm, a clustering  $\mathcal{C}_{init}$  is generated by Algorithm 2, a greedy heuristics reported in our previous work [3]. The solution  $\mathcal{C}_{init}$  is considered as the first candidate solution. This candidate solution is iteratively modified using the operations of genetic algorithm meta-heuristic in the search for a better solution. This algorithm stops when the current best solution cannot be further improved for a number of  $t$  consecutive steps. The algorithm performs the following tasks iteratively:

1. Measure the fitness ( $f_i$ ) of each cluster ( $C_i$ ) for all clusters in the solution
2. Create cluster pairs ( $p$ ) based on the fitness
3. Perform cross-over between the clusters  $C_a$  and  $C_b$  in a pair  $p_i = (a, b)$  by exchanging member vertices of the clusters, for all pairs  $p \in P$
4. Perform mutation by changing the order of vertices within a randomly picked cluster

5. Compare the  $\kappa$  of the candidate solution in hand with the best solution found so far. If  $\kappa$  for candidate solution is better, change the best solution to the candidate solution

---

**Algorithm 3** FitnessCalculation

---

**Input:** Clustering  $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$ , Call Graph  $G(V, E)$   
**Output:** List of fitness  $F = f_1, f_2, \dots, f_m$  where  $f_i$  is the fitness of cluster  $C_i$

```

1: Begin
2: for each  $C_i \in \mathcal{C}$  do
3:    $n \leftarrow 0$ 
4:   for each pair  $v_j, v_u \in C_i$  do
5:     if  $(v_j, v_u) \in E$  then
6:        $n \leftarrow n + 1$ 
7:     end if
8:   end for
9:   Compute  $\chi_{C_i}$  using Eq. 3
10:   $f_i \leftarrow n + \chi_{C_i}$ 
11: end for
12: End

```

---



---

**Algorithm 4** ParentSelection

---

**Input:** Clustering  $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$ , Fitness list  $F = f_1, f_2, \dots, f_m$   
**Output:** List of cluster index pairs  $\mathcal{P} = \{p_1, p_2, \dots, p_{\lceil \frac{m}{2} \rceil}\}$  where  $p_i = \{\mathbb{N}, \mathbb{N}\}$

```

1: Begin
2: Order  $C_i \in \mathcal{C}$  in order of  $f_i$  associated with  $C_i \quad \forall i=1,2,\dots,m$  to produce list  $\mathcal{C}_{ordered}$ 
3: List  $\mathcal{P} \leftarrow \emptyset$ 
4:  $i \leftarrow 0$ 
5: for each  $C_i, C_{i+1} \in \mathcal{C}_{ordered}, i = 1, 2, \dots, |\mathcal{C}_{ordered}| - 1$  in order of  $\mathcal{C}_{ordered}$  do
6:   add  $(i, i + 1)$  to list  $\mathcal{P}$ 
7:    $i \leftarrow i + 2$ 
8: end for
9: End

```

---

The steps involving fitness calculation and parent selection is illustrated in Algorithm 3 and 4. Two measures have been used for fitness calculation that includes the number of intra-cluster edge and characteristic path length ( $\chi$ ). The fitness of each cluster is realized as the summation of those two measures. The result of this step is a fitness list ( $F$ ) which works as the basis of Parent Selection. This step involves sorting the fitness list in descending order and pairing the clusters, corresponding to a fitness value, from top to bottom. It produces a list of cluster pairs  $P = \{p_1, \dots, p_{\lfloor m/2 \rfloor}\}$  for cross-over.

Algorithm 5 and 6 illustrates the procedure involving cross-over and mutation. A random number  $\tilde{o}(v)$  is assigned to each vertex  $v \in V$ , which is used in mutation and cross-over operations. The cross-over operation uses this order to select vertices to be exchanged between clusters. The mutation operation exchanges the order of two vertices within a cluster. Cross-over will take place for each pair that was generated during Parent Selection. After completing the cross-over, mutation takes place. However, this will occur based on a probability distribution. The process involves shuffling the order of two vertices from a randomly picked cluster.

## 4 Experimental Results

Our proposed genetic algorithm has been implemented using C++ programming language on a 32-bit Ubuntu 12.04 Operating System, 2.1 GHz Dual Core processor, 1 GB RAM computer.

Three different datasets used in [3] have been used to experiment with our proposed genetic algorithm. *BTF*, *RBlo* were generated from two different scientific software and *Synthetic166* was synthetically generated. Table 1 describes the data set in terms of the number of user defined functions and function calls.

Table 2 presents the number of clusters generated by our proposed algorithm in contrast with the algorithms in [3]. Our proposed algorithm does not change the number of clusters from the initial seed, it just enhances the

---

**Algorithm 5** CrossOver

---

**Input:** Clustering  $\mathcal{C}$ , List of cluster index pairs  $\mathcal{P} = \{p_1, p_2, \dots, p_{\lceil \frac{m}{2} \rceil}\}$ , order  $\tilde{o}$  of  $v \in V$

**Output:** Clustering  $\mathcal{C}$

```
1: Begin
2: for each  $p_i \in \mathcal{P}$  do
3:    $\mathcal{C}' \leftarrow \mathcal{C}$ 
4:   Generate random number  $r_1$  and  $r_2$  such that  $1 < r_1 < |\mathcal{C}_a|$ ,  $1 < r_2 < |\mathcal{C}_b|$ ,  $\{a, b\} \in p_i$ 
5:    $\mathcal{C}_{a_{tmp}} \leftarrow \mathcal{C}_a$ 
6:    $\mathcal{C}_{b_{tmp}} \leftarrow \mathcal{C}_b$ 
7:   for  $i \in 1 : r_1$  do
8:     Pick  $v_i \in \mathcal{C}_{a_{tmp}}$  in desc. order of  $\tilde{o}(v_i) \in \mathcal{C}_{a_{tmp}}$ 
9:      $\alpha \leftarrow \alpha \cup v_i$ 
10:     $\mathcal{C}_{a_{tmp}} \leftarrow \mathcal{C}_{a_{tmp}} \setminus v_i$ 
11:   end for
12:   for  $j \in 1 : r_2$  do
13:     Pick  $v_j \in \mathcal{C}_{b_{tmp}}$  in desc. order of  $\tilde{o}(v_j) \in \mathcal{C}_{b_{tmp}}$ 
14:      $\beta \leftarrow \beta \cup v_j$ 
15:      $\mathcal{C}_{b_{tmp}} \leftarrow \mathcal{C}_{b_{tmp}} \setminus v_j$ 
16:   end for
17:    $\mathcal{C}'_a \leftarrow \mathcal{C}_{a_{tmp}} \cup \beta$ 
18:    $\mathcal{C}'_b \leftarrow \mathcal{C}_{b_{tmp}} \cup \alpha$ 
19:    $\mathcal{C}' \leftarrow \mathcal{C} \setminus \mathcal{C}_a$ 
20:    $\mathcal{C}' \leftarrow \mathcal{C} \setminus \mathcal{C}_b$ 
21:    $\mathcal{C}' \leftarrow \mathcal{C} \cup \mathcal{C}'_a \cup \mathcal{C}'_b$ 
22:    $\mathcal{C}' \leftarrow \mathcal{C} \cup \mathcal{C}'_b$ 
23: end for
24:  $\mathcal{C} \leftarrow \mathcal{C}'$ 
25: End
```

---

---

**Algorithm 6** Mutation

---

**Input:** Clustering  $\mathcal{C}$ , Order  $\tilde{O}$  of  $v \in V$

**Output:** Clustering  $\mathcal{C}$

```
1: Begin
2:  $\mathcal{C}' \leftarrow \mathcal{C}$ 
3: Generate a random number  $r \in \mathbb{R}$  between 0 to 1
4: if  $r \leq \epsilon$  then
5:   Randomly pick  $C \in \mathcal{C}$ 
6:   Randomly pick  $v_a, v_b \in c$ 
7:    $tmp \leftarrow \tilde{o}(v_a)$ 
8:    $\tilde{O}(v_a) \leftarrow \tilde{o}(v_b)$ 
9:    $\tilde{o}(v_b) \leftarrow tmp$ 
10: end if
11:  $\mathcal{C} \leftarrow \mathcal{C}'$ 
12: End
```

---

Table 1: Number of Vertices and Edges of experimental dataset

Dataset	Number of Vertices	Number of edges
BTF	14	31
RBLo	61	372
Synthetic166	166	450

Table 2: Number of clusters produced by proposed heuristics

Dataset	Monte Carlo	Greedy	Genetic
BTF	3	4	4
RBLo	7	8	8
Synthetic166	23	13	13

Table 3: Performance of Genetic Algorithm on Different Datasets

Dataset	CC( $\Psi$ )		CPL( $\chi$ )		Kal( $\kappa$ )	
	Seed	Final	Seed	Final	Seed	Final
BTF	0.308333	0.315972	1.20536	0.794444	5	9
RBlo	0.474293	0.521471	1.52528	0.378843	5	77
Synthetic166	0.333908	0.390482	4.10824	3.75334	35	135

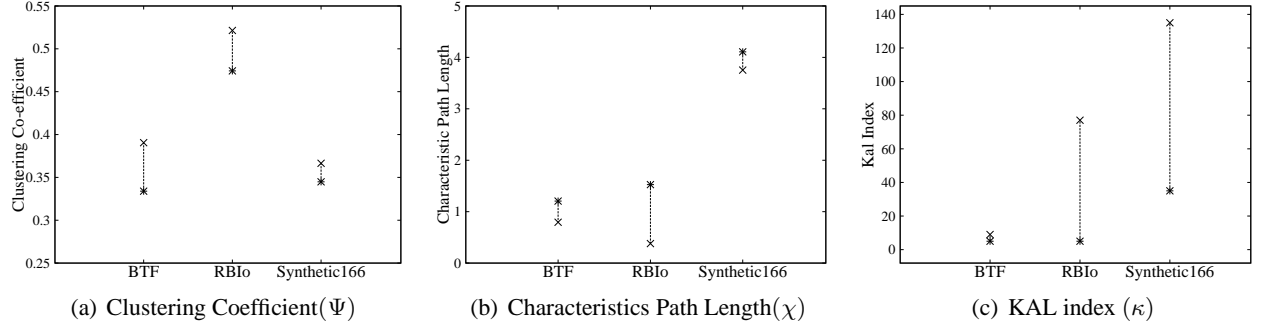


Figure 1: Genetic Algorithm on *BTF*, *RBlo* and *synthetic166*

solution quality. Using the proposed algorithm 4, 8, and 13 clusters were obtained for dataset *BTF*, *RBlo*, and *Synthetic166* respectively.

Results obtained by applying Algorithm 1 on the datasets are presented in Table 3. The scores for CC( $\Psi$ ), CPL( $\chi$ ) and Kal( $\kappa$ ) have been improved by 0.097, 0.39 and 5.66 times respectively using our proposed algorithm. Figure 1 presents a graphical representation of data in Table 3, here star (\*) symbol refers to the score of the initial seed and cross (x) symbol represent result obtained by our proposed algorithm.

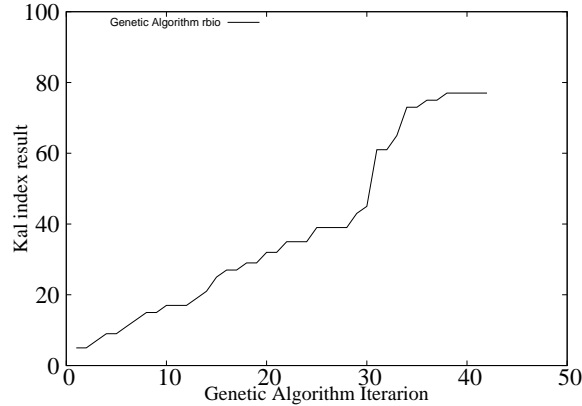


Figure 2: Iteration of Genetic Algorithms on *RBlo*

Figure 2 presents the gradual improvement done by our proposed algorithm on dataset *RBlo*. The  $\kappa$ -index score of initial seed was 5, it improved over the next 43 iterations of the algorithm to a score of 77. This score, since did not improve over the next 5 iterations, has been reported as the best solution obtained.

Figure 3 presents the comparison of average scores of Monte Carlo based algorithms and scores of Greedy approaches [3] with the scores obtained by proposed algorithm results. Sign (\*), (x), and (+) denotes the Kal( $\kappa$ ) index of Monte Carlo, Greedy, and Genetic Algorithm of dataset *RBlo*. This figure indicates that our proposed genetic algorithm produces significantly better result than the Monte Carlo and greedy algorithms.

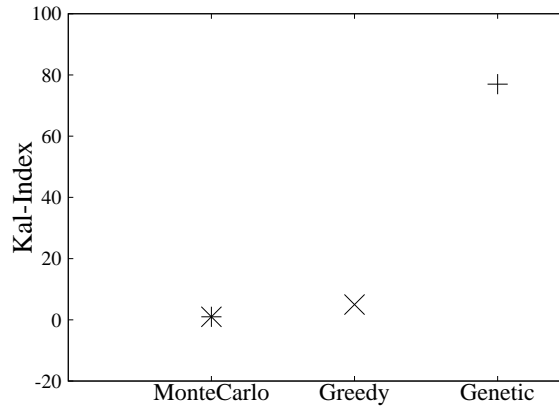


Figure 3: Comparison of  $Kal(\kappa)$  on *RBIO*

## 5 Conclusion

This paper addressed a design migration problem from Structured Language to Object Orient Paradigm. Here, we proposed a Genetic Algorithm based meta-heuristic approach and presented the test result on datasets reported in [3]. Our proposed approach achieved 40% improvement compared to greedy algorithms and 49.5% improvement compared to the Monte Carlo approaches presented in [3].

In future we are interested to enhance the performance of the proposed algorithm trying variations of the *FitnessCalculation* and *ParentSelection* functions. The *ParentSelection* function in our proposed algorithm selects pairs of clusters in order of fitness. Thus two high fitness clusters are crossed over. We identify this as a potential area of improvement, where we want to cross over the low fitness clusters with the high ones to see all clusters have a high fitness. The *FitnessCalculation* function may be enhanced inspired by the  $\kappa$  index, which to our understanding, is the most suitable matrix to measure the strength of a clustering scheme.

Currently, our research group is working towards developing a local search based algorithm to find an approximate solution to the problem. We are also interested in developing an Ant Colony Optimization based meta-heuristic approach for the problem. It would be great to be able to validate OOP design clue generated by the algorithms by practicing OOP professionals.

**Acknowledgements:** This research was conducted by Optimization Research Group of Institute of Information Technology, University of Dhaka.

Our sincere gratitude to Dr. Shahadat Hossain, Associate Professor, Dept. Math & Computer Science, Univ. of Lethbridge, AB, Canada for presenting this problem to us and thanks to Mr. Ahmed Tahsin Zulkernaine for providing sample datasets.

### References:

- [1] K. Bennett, "Legacy systems: coping with stress," *Software, IEEE*, vol. 12, no. 1, pp. 19–23, 1995.
- [2] B. G. Ryder, "Constructing the call graph of a program," *Software Engineering, IEEE Transactions on*, no. 3, pp. 216–226, 1979.
- [3] S. Siddik, A. U. Gias, and S. M. Khaled, "Optimizing software design migration from structured programming to object oriented paradigm," in *16th International Conference on Computer and Information Technology*, (Khulna University, Bangladesh), IEEE, December 2013. Accepted.
- [4] M. Selim, "A genetic algorithm for design migration from structured language to object oriented paradigm," tech. rep., Institute of Information Technology, University of Dhaka, Bangladesh, 2013.
- [5] H. M. Sneed and E. Nyary, "Extracting object-oriented specification from procedurally oriented programs," in *2nd Working Conference on Reverse Engineering*, (Toronto, Ont., Canada), pp. 217–226, IEEE, 1995.
- [6] O. Maqbool and H. A. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.

- [7] M. A. Heroux and J. M. Willenbring, "Barely sufficient software engineering: 10 practices to improve your cse software," in *Software Engineering for Computational Science and Engineering, 2009. SECSE'09. ICSE Workshop on*, pp. 15–21, IEEE, 2009.
- [8] V. Dineshkumar and J. Deepika, "Code to design migration from structured to object oriented paradigm," *International Journal of Information and Communication Technology Research*, vol. 1, 2011.
- [9] P. Fränti, J. Kivijärvi, T. Kaukoranta, and O. Nevalainen, "Genetic algorithms for large-scale clustering problems," *The Computer Journal*, vol. 40, no. 9, pp. 547–554, 1997.
- [10] J. Kivijärvi, P. Fränti, and O. Nevalainen, "Self-adaptive genetic algorithm for clustering," *Journal of Heuristics*, vol. 9, no. 2, pp. 113–129, 2003.
- [11] Y. Wang, "Fuzzy clustering analysis by using genetic algorithm," *ICIC Express Letters*, vol. 2, no. 4, pp. 331–337, 2008.
- [12] E. R. Hruschka and N. F. Ebecken, "A genetic algorithm for cluster analysis," *Intelligent Data Analysis*, vol. 7, no. 1, pp. 15–25, 2003.
- [13] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [14] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.
- [15] D. Braha and Y. Bar-Yam, "The statistical mechanics of complex product development: empirical and analytical results," *Management Science*, vol. 53, no. 7, pp. 1127–1145, 2007.